EECS 3201 - Lab #5 Report
Registers, Sequential Simulation and Measurement
Group 04 Section B:
Ajay Deonandan #217987215
Isaiah Gocool #218918052

Overview

Objective

In lab 5 part A, we were given three different counters that we needed to implement. These counters would run using either a fast or slow clock based on what the value of SW[9] is. SW[1:0] are used to select which counter would be used, and then the circuit can be analyzed using RTL Viewer, ModelSim, and SignalTap. For part B, we had to create a 5-bit linear feedback shift register with an asynchronous preset and a parallel load capability. The behaviour of this circuit would be tested using ModelSim or SignalTap to see if the counters still work correctly.

Acronyms

The 10 DE10-Lite toggle switches are referred to as SW[N] where N is the number of the switch being referred to. The LEDs on the board are referred to as LEDR[N], where N is an integer from 0 to 7 that represents which LED is selected. KEY[N] represents the two buttons on the board where N is the button number.

Design Approach

Inputs and Outputs

On the DE10 Lite board, LEDR[0:7] were used to display the bits of each number for the counter. For example, if the counter outputs the number 3, then LEDR[0] and LEDR[1] will be turned on because the binary representation of 3 is 4'b0011. KEY[0] was used as the button to reset the clock, SW[9] was used to change between the fast and slow clock speeds, SW[1:0] was used to swap between using the three counters and MAX10_CLK1_50 was used as our ClockDivider clock.

Counter

Our ClockDivider module utilized the DE-10's clock 1 which provides a 50MHz clock input. A multiplexer was used to select whether the slow or fast clock speed should be used, based on the input of SW9. If SW9 was on, cout would be toggled every 50 cycles for the fast clock, and if it was off then cout would be toggled 5 million cycles for the slow clock. In our "topmod" module, we initialized all the counters with three different count registers to represent each counter. An always@ block was used to check every time the clock incremented, and then LEDR[7:0] would be assigned values based on what the value of the current counter is determined by the states of SW[1:0].

RTL Viewer

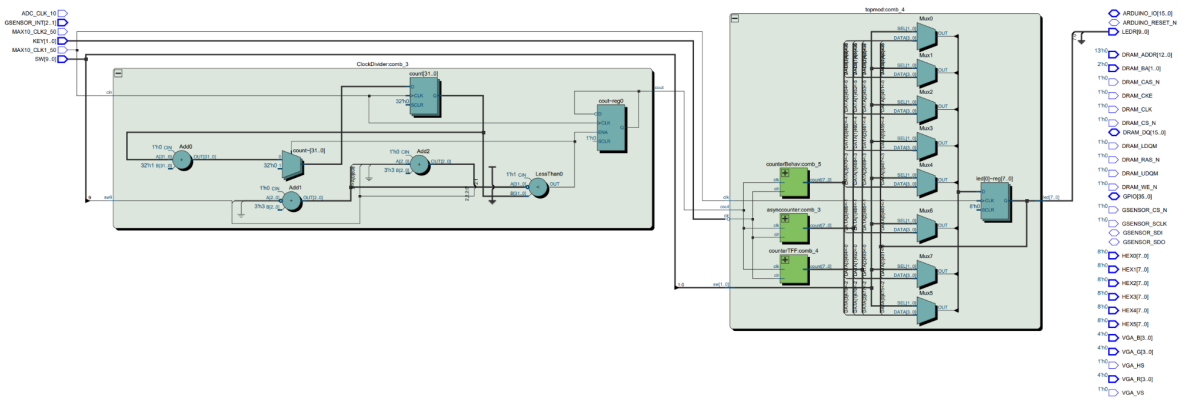

Figure 1: Circuit diagram of our top level module, which instantiates both the ClockDivider and the 3 counters.

When we used the RTL Viewer, it gave us a circuit diagram view of the program we created. This diagram followed our logic well and was a lot more complicated than we expected it to be. The circuit diagram of the ClockDivider was what we expected it to be since the code was short and easy to follow. However, the diagrams for the counters were far more complex.

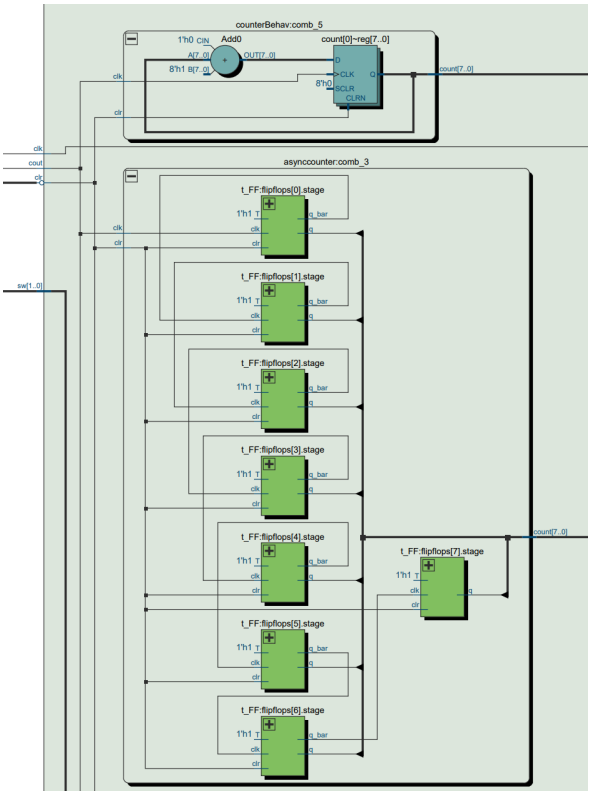

Figure 2: RTL Viewer of counterBehav and asynccounter.

The circuit diagrams of counterBehav and asyncccounter were close to what was expected, but we were surprised with the amount of flip-flops needed for asynccounter. This makes sense given that a for loop was used to generate different flip-flops, but as a counter it's more complex than counterBehav.

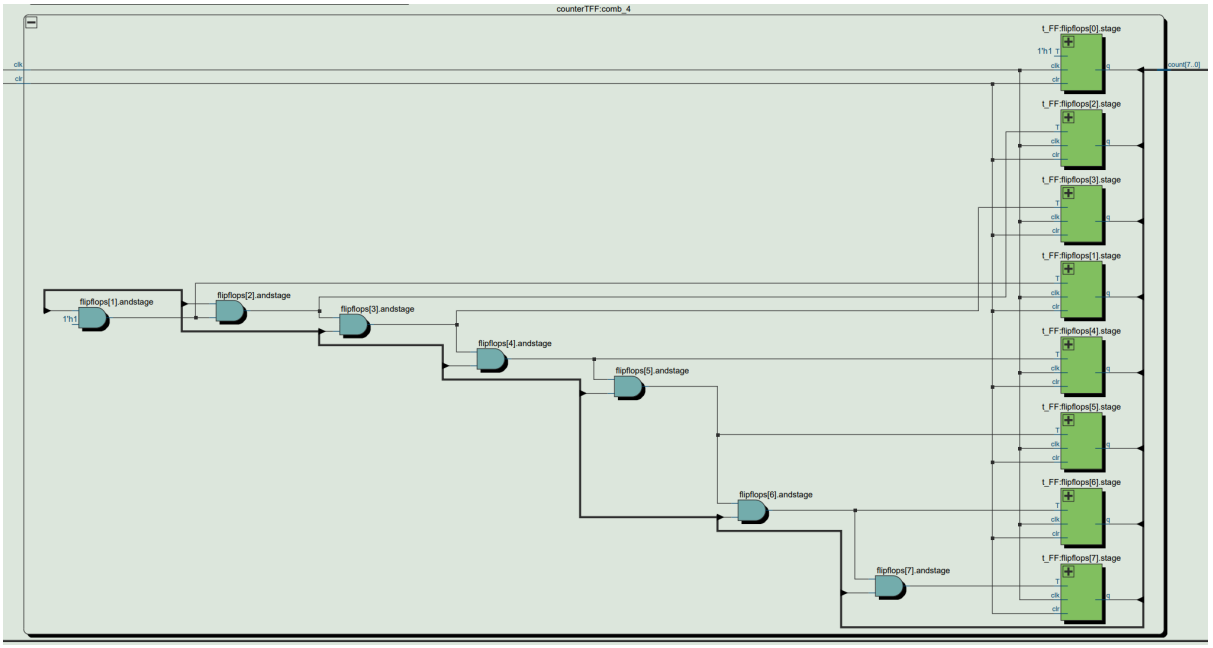

Figure 3: RTLViewer of counterTFF.

counterTFF had the most surprising circuit due to the amount of AND gates used.

RTL Simulation

The following images show the timing diagrams for each counter that were created using ModelSim's RTL simulator. The clock and clear were both used as inputs for the counters, and then the rest of the variables were calculated based on their values.

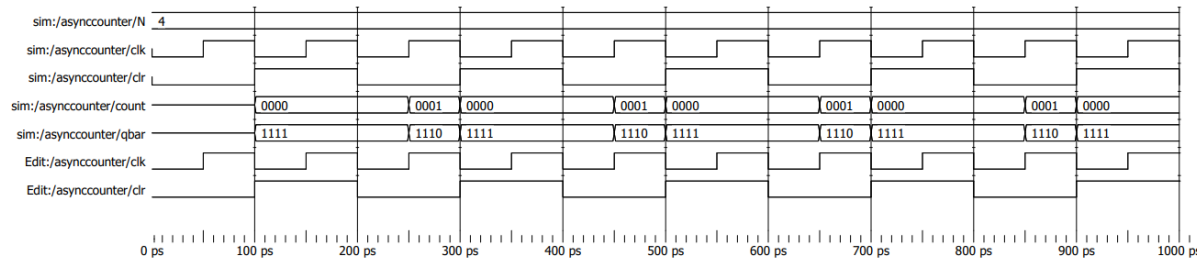

Figure 4: RTL simulation wave graph of asynccounter.

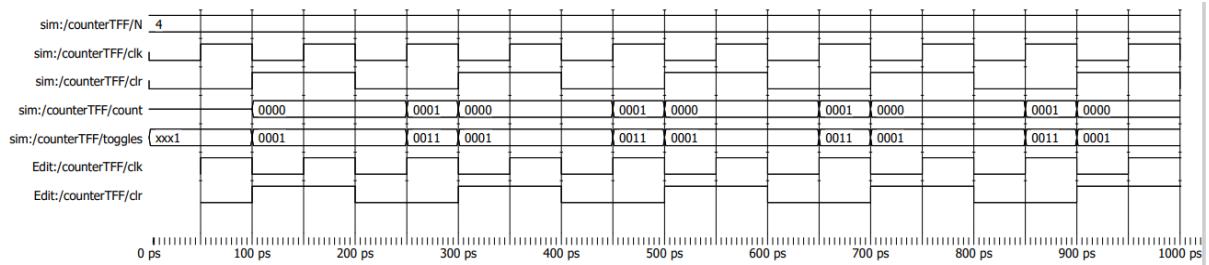

Figure 5: RTL simulation wave graph of counterTFF.

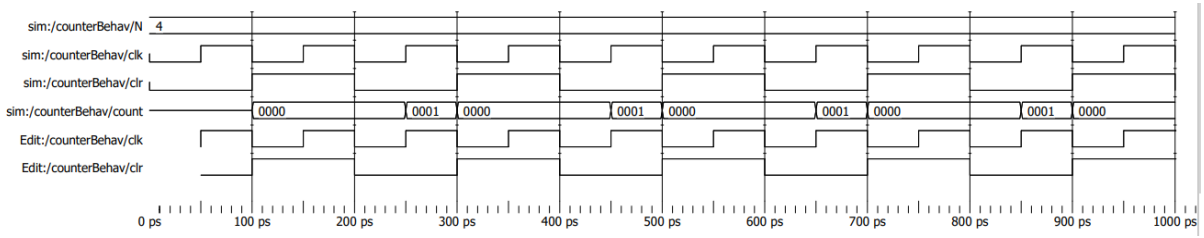

Figure 6: RTL simulation wave graph of counterBehav.

The behaviour of the wave graphs followed our expectations. Each counter had the same outputs, but they used different methods to achieve that result, which was explained in the ‘Counters’ section of ‘Design Approach’.

SignalTap Simulation

The following images capture the behavior of the three counters utilizing SignalTap. The counters were run using the fast clock that was previously set up, having a sample depth of 8K and having a trigger condition of the counters counting up to FF (1111 1111). We also utilized an additional trigger condition of KEY[1] being pressed (set to low) which was used to combat an issue that was being encountered. We go into more detail about this issue in the ‘issues encountered’ section of this report. The exported timing diagrams of our circuit’s behavior under a fast clock:

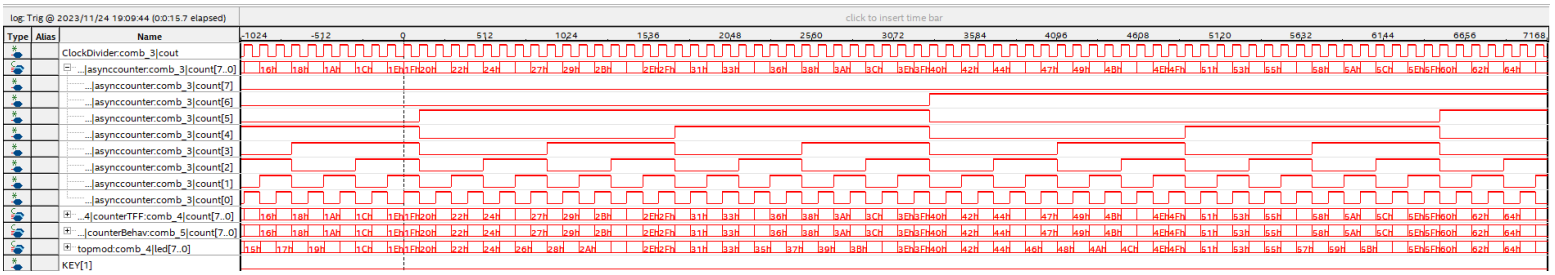

Figure 7: SignalTap graph of asynccounter.

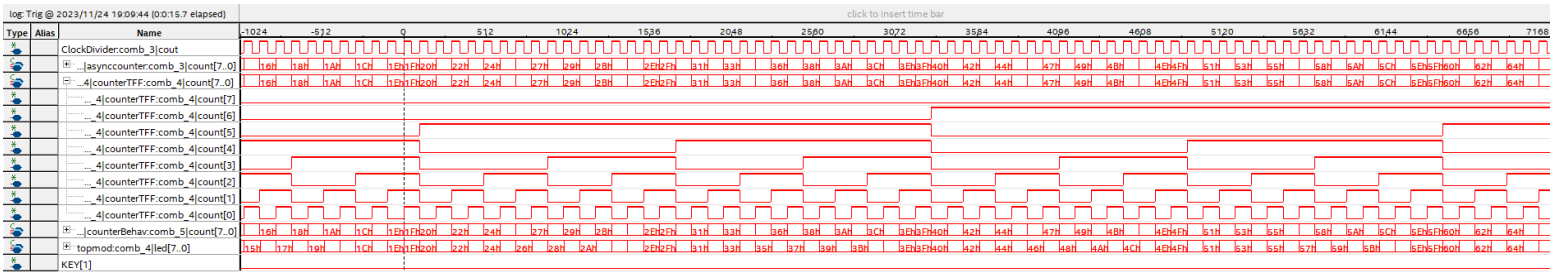

Figure 8: SignalTap graph of counterTFF.

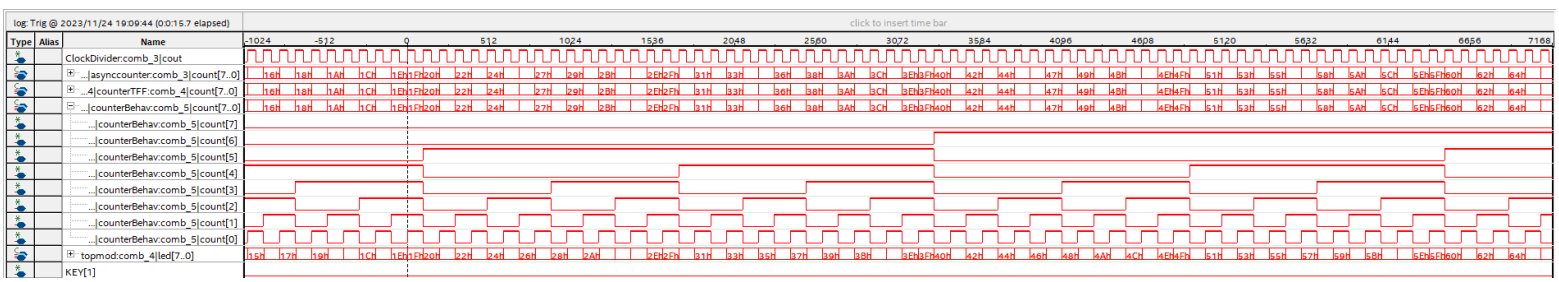

Figure 9: SignalTap graph of counterBehav.

The expected relationship between counter outputs can be seen in each graph. When looking at the graphs, the timing diagram of an 8 bit binary counter can be seen. When looking at the time scale, we can see that over the course of a certain period of time, the counter bits for each counter increment as a binary counter for each positive edge of our input clock. Count[0] to Count[7] for each counter changes its state to represent the binary count until the counter reaches FF. (all bits in the counter are high). Everytime a new bit in each counter changes state, it is clear how the other bits in the counter change at that certain point of time based on it in the timing diagram. Once the final bit (count[0]) gets set to high when the other bits (count[1]-count[7]) are all high, the trigger value is reached and the timing diagram executes. When comparing the three counters with each other, they all show the same expected relationship between their output bits.

Test Strategy

To start, we implemented the ClockDivider module included in the fourth lab manual, and changed the cycle time so that it fit the requirements of this lab (slow and fast clock speeds). After that, we instantiated each counter in our module called topmod, and used if statements to check which counter should be used based on the inputs of SW[1:0]. To test if it worked, we blasted the program onto the board and checked if the correct bits were displayed through the LEDs. When we were conducting the simulations, we tried changing which inputs were used and the clock speeds for them to see how the outcomes would change.

To test our completed lab, we utilized tools such as Model Sim and SignalTap. This let us verify that each counter was working as expected. When testing our lab physically, we started with first using two slower clock speeds to check to see if one of our counters worked (we started with the asynchronous counter). Rather than running all three counters simultaneously, we first wanted to see if each counter ran by themselves. In order to do this, we slowed down our fast and slow clock speeds 0.5s and 1s so we could visually see the changes in the bits when the counters worked. This also allowed us to check to see if SW9 was properly changing clock speeds between our slow and fast clocks. If we used the clock speeds mentioned in the lab manual, we wouldn't be able to see any changes as the clock speeds are too fast to visually notice anything changing. After we tested each counter to make sure they worked without our topmod module, we then implemented them into our topmod and tested them again simultaneously using SW[1:0] to swap between them. We kept our clock slower so we could still see any visual changes in the counters. Once we made sure our counters were working simultaneously and nothing was working when no switches in SW[1:0] were turned on, we then went to ModelSim and SignalTap to look at the signal changes at a deeper level.

Issues Encountered

An issue we encountered was our inability to view a complete timing diagram when using SignalTap. We would set the appropriate trigger value; however, once the acquisition process was completed on SignalTap, it would only show the instance before the trigger value before getting to the trigger value. If our trigger value was 1F, it would only show the counter at 1E before getting to 1F. We tried various approaches to fix this issue - some include using different clock speeds (slower clocks) and forcing our counter to stay at 0 by holding the reset button before the acquisition process began on SignalTap, none of these succeeded however. We even approached other groups in our lab about the issue, many were having a similar problem to us. We even approached our sections TA about the issue and they were also unable to solve the problem as to why SignalTap was behaving this way. We then tried using another trigger value on top of the trigger value of stopping at FF - this additional trigger was the unused button (KEY[1]). We would wait for a complete cycle to occur on

our counter and then hold down KEY[1] on the second occurrence of our timer. This served as a fix to capture the signal changes of our counter from 0 to FF.

Another issue we had was getting the RTL simulation to work. We were initially confused with how to choose what the inputs and outputs would be, so we tried different combinations and saw what worked. After intuitively thinking about it, we only set the inputs of the module as the inputs of the wave, and left everything else as is. We thought that the wires used in the module would need to be instantiated as inputs, but then we realized that only the actual inputs of the module needed to be instantiated. We gave the inputs set clock values and then ran the simulation to get the timing diagrams of the rest of the outputs/wires.

For part B of the lab, we created our 5-bit linear feedback shift register circuit based on the design for a 3-bit linear feedback shift register given to us in the textbook. After that, we tried to instantiate it so that it would shift the bits of the counter. The main problem we had was that we didn't know what inputs we should pass into the module for the shift, load, and preset functions. Ultimately, we were unable to solve this problem so part B was left unfinished.